운영체제 03분반

실습 6회차

201402447 한원희

```
/*
 * To-do : initialize for game
 * get named semaphore, fifo, etc.
 */

mkfifo("pingpong.txt", 0666);              // fifo 파일을 생성해줍니다
fd = open("pingpong.txt", O_RDWR|O_CREAT, S_IRWXU);  // 파일 오픈

sem_unlink("pingongsema");
if((sem = sem_open("pingongsema", O_CREAT, 0644, 1)) == SEM_FAILED){
            perror("open");
            exit(1);
}
```

Server.c 파일입니다. 먼저 fifo 파일을 생성해주고 열어줍니다. 그 이후 semaphore을 오픈합니다. 문법은 pdf파일을 참고하였습니다.

```
 * To-do : Round1 ping
 * without [opponent] string
 */

printf("Your turn!₩n");
memset(buf, 0x00, BUF_SIZE);
fgets(buf, BUF_SIZE, stdin);              // 먼저 입력을 받아줍니다

if (strcmp(buf, pingstr)){                // ping이 아니라면 호출
            printf("wrong! -20₩n");
            score -= 20;
}

write(fd, buf, strlen(buf));              // 파일에 작성

sem_post(sem);                            // 카운트 값 1 증가
sleep(1);
```

Round1을 미리 지정해줍니다. 입력을 받고 그 이후 strcmp 함수를 이용해 검사해줍니다. pingstr에는 ping이 저장되어 있습니다. Strcmp는 두 문자열이 같을때만 0을 출력하므로 다르다면 if문 안을 실행합니다. 그 이후 fifo 파일에 작성해주고 semaphore의 값을 1 증가시켜 줍니다.

```
for (cnt=1; cnt<5; cnt++)
{
        sem_wait(sem);              // 카운트 1 감소시키고 0이 됐으니 대기

        read(fd, buf, BUF_SIZE);    // 파일을 읽어 buf에 저장

        printf("[opponent] %s", buf); // buf 출력
        printf("Your turn!₩n");
        memset(buf, 0x00, BUF_SIZE);
        /*
         * To-do : complete game process
         */
        fgets(buf, BUF_SIZE, stdin);   // 입력

        if (strcmp(buf, pingstr))
        {
                printf("wrong! -20₩n");
                score -= 20;
        }

        write(fd, buf, strlen(buf));    // 파일에 작성

        sem_post(sem);              // 카운트 1 증가
        sleep(1);
}
```

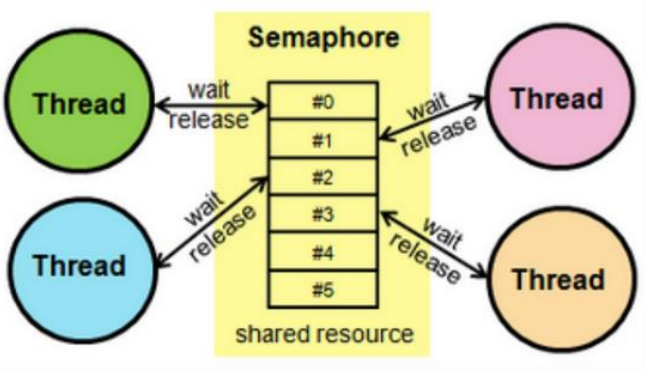
2라운드 때부터 해당하는 장소입니다. Sem_wait으로 카운트를 0으로 만들고 대기시킵니다. fifo파일을 읽어 들인 뒤 opponent 를 출력해줍니다.

그 이후 round1과 같이 입력을 받고 검사하고 fifo에 작성해 준 이후 sem_post로 카운트를 증가시켜 줍니다.

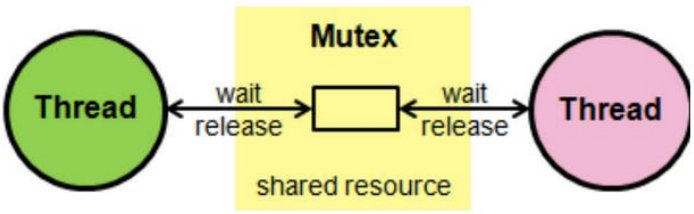
Client.c도 많이 다르지 않습니다.

결과입니다.

2.Mutex와 Semaphore의 차이



Semaphore – signaling mechanism, 현재 공유자원에 접근할 수 있는 쓰레드, 프로세스의 수를 나타내는 값을 두어 상호 배제를 달성.



Mutex – 한 쓰레드, 프로세스에 의해 소유될 수 있는 key를 기반으로 한 상호 배제.

차이점

Semaphore는 Mutex가 될 수 있지만, Mutex는 Semaphore가 될 수 없다

- Mutex는 상태가 0, 1 두개 뿐인 binary semaphore입니다.
  Semaphore는 여러 쓰레드, 프로세스들이 쓸 수 있지만 Mutex는 하나로 고정되어 있습니다.

Semaphore는 현재 수행중인 프로세스가 아닌 다른 프로세스가 semaphore를 해제할 수 있지만 Mutex는 lock을 획득한 프로세스가 반드시 그 lock을 해제해 주어야 합니다.

Semaphore는 시스템 범위에 걸쳐있고 파일 시스템상의 파일 형태로 존재합니다. 반면 Mutex는 프로세스 범위를 가지며 프로세스가 종료될 때 자동으로 clean up 됩니다.